Introduction

This book contains 35 BASIC programs for the Commodore 64, each with easy-to-follow directions, a complete program listing, notes about important variables, and an outline of how the program works. You don't need to know how to program the Commodore 64 to use this book—just type the program into your computer, **SAVE** it on disk or cassette tape, and then type **RUN**. If you already know BASIC, the information about important variables and how the programs work will help you understand how to modify the programs.

Our PROOF-IT proofreading program helps you avoid typing errors. With PROOF-IT, you avoid the frustrating "debugging" needed when typing errors keep programs from working right. Another helpful feature is the special notation used in the program listings. The notation shows each key you need to press, and gives a count for repeated strings of characters. Commodore character graphics are hard to read with "normal" listings on a Commodore printer. Finally, all programs (except the "QUICK" series) are *identical* from line 60000 on. This "standard framework" handles keyboard input, joystick manipulation, the title screen, checking for **Q** to quit, and resetting the C-64 to standard colors, among other things. The standard framework saves lots of typing, since you type it only once (using PROOF-IT to make sure it's correct), **SAVE** it on disk or tape, and then **LOAD** it before typing a program from the book.

THE PROOF-IT PROOFREADER

PROOF-IT is a powerful proofreading program that helps you avoid typing mistakes. As you type each line of a program, PROOF-IT calculates a "proof number" which it displays at the top of the C-64 screen when you press RETURN. At the end of each program line in the listings, we print the correct proof number for that line. If the number PROOF-IT shows matches the number in the book, the line you just typed is correct. If the numbers are different, you need to carefully check your line against ours, and correct the error. You may have misunderstood our listing notation (did you forget to hold down the SHIFT or COMMODORE keys?). Other common mistakes include using the letter "l" (el) instead of the number "1," the letter "O" (oh) rather than "0" (zero), or typing the wrong number of characters.

PROOF-IT helps you type programs from this book that work the first time. It catches almost all typing mistakes, including transposed characters, such as "XT" when you meant "TX." PROOF-IT checks on "important" spaces inside quotes but ignores "unimportant" spaces outside of quotes. It even lets you use the handy "shorthand" of BASIC keyword abbreviations, so you can type a question mark instead of the keyword "PRINT."

Getting Started with PROOF-IT

1. Type **NEW**.
2. Type PROOF-IT *exactly* as shown. Be careful to include blanks, since they are important.
3. **SAVE** the program on disk or tape. *Warning:* you'll lose PROOF-IT if you do not **SAVE** it now!
4. Type **RUN**. The copyright notice will appear, then the message "SETTING UP" followed by a series of dots.
5. If everything is correct, the C-64 will print "READY." When you type a BASIC line and press RETURN, the proof number will appear in the upper left-hand corner of your screen in reverse video.

PROOF-IT is great, but before you can use it you must type it into your C-64. Unfortunately, you won't have a smart program (such as PROOF-IT) to help you locate and correct typing errors.

To help solve this "getting started" problem, PROOF-IT checks itself four different ways, and prints an error message and stops if it finds

anything wrong. First, it first checks the crucial numbers on line 999. Next, it checks the main BASIC section for errors. If everything looks good, it reads a DATA line and checks it. If a DATA line is bad, it stops and reports the bad line. Finally, it counts the number of DATA lines, and complains if there are too few or too many. All this checking takes a few seconds, but it's worth it, since PROOF-IT must be correct so it can accurately check all the programs you type from the book.

Since getting PROOF-IT and the "Standard Framework" correct is so vital, you can get these important programs on disk or tape for a nominal charge. See the section *Obtaining Programs on Disk* for details.

Using PROOF-IT

Once you've successfully typed PROOF-IT, **SAVE**d it on disk or tape, and it has **RUN** correctly, you are ready to type a program. If you are anxious to get started, you may want to select one of the short QUICK programs, since you don't have to type the standard framework first. If you type anything other than one of the QUICK programs, your next step is to type the standard framework (see listing on page xv) and **SAVE** it for future use.

If you are typing one of the QUICK programs, first **LOAD** the PROOF-IT program and **RUN** it. When it says "READY," type the first line from the listing you have selected. When you press RETURN, check the proof number on the screen with the number printed at the end of each line in this book.

At the end of each program line in the book the proof number is shown as a REMark statement. For example, in this line,

```
150 PRINT "HELLO" :rem 6470
```

the part ":rem 6470" is the proof number. Do *not* include the trailing colon, "rem" and proof number when you type a line, since they are not part of the program. When the proof number shown on the screen matches the number in the book, the line has been typed correctly. If the proof numbers differ, check the line carefully, correct the error, press RETURN again, and then again check the proof number on your screen with the proof number shown in the listing.

Lines 3–10 in the program listings do not have proof numbers in the book. Those lines don't change how the program runs, and eliminating the proof numbers makes the copyright notice and version date easier to read. However, you do need to type lines 3–10 so the proof number and line count for the entire program will check correctly with the numbers shown at the end of the listings in the book.

Using "P" to Check a Whole Program

In addition to showing the proof number for each line, PROOF-IT can also check an entire program to see that you have typed the right number of lines, and the right line numbers. With PROOF-IT active, if you type the command **P** on a line by itself, and press RETURN, PROOF-IT will tell you the number of lines in the program, and the proof number for all the line numbers in the program. The correct number of lines and program proof value are printed at the end of each listing.

If the numbers shown by PROOF-IT when you press **P** (and RETURN) match the numbers at the end of the listing, you have typed all the lines, and they all have the correct line numbers. Should the proof numbers not match, you'll know whether to look for either missing or extra lines. (Hint: did you forget to include the standard framework?) PROOF-IT doesn't tell you which lines are missing or extra, but you can list the program a screen at a time and check just the line numbers against the book.

The **P** command checks all the line numbers in the program, but does not check the rest of the program. The line-by-line proof numbers are your assurance that each line is correct, while the **P** values tell you that you typed the right number of lines, and that you got all the line numbers correct.

You can go back and check previous lines at any time with PROOF-IT, without retyping the line. **LIST** the line or lines you want to check, and use the cursor keys to put the cursor anywhere on the line you want to check. When you press RETURN, the proof number for that line will be shown.

Remember, PROOF-IT calculates a proof number for each line and prints it on the screen. But it's up to you to check that number against the correct proof value printed at the end of each line in the book. If you have trouble getting a program to work, **LIST** it in groups of about 15 lines, move the cursor to each line and press RETURN. Check each proof number against the number in the book. (It helps to have a partner, so one person works the C-64

keyboard, and the other reads the correct proof values from the book.)

PROOF-IT can be "turned off" by typing the letter **Q** followed by a RETURN on a line by itself. This "deactivates" proofing, but leaves the program intact. It can be turned back on by giving the command **SYS 51000**.

PROOF-IT will help you type correct programs, as long as you take the time to compare the proof number for each line with the book. The **P** command checks whether you have typed all the lines. Don't forget that the values shown at the end of each listing include the standard framework for all programs other than the QUICK series.

```
  1 PRINT CHR$(147);"PROOF-IT"
  5 PRINT "(C)␣1984␣THE␣CODE␣WORKS"
 10 REM AS OF 08AUG84
 20 REM BY MIKE HOWARD
100 PRINT:PRINT "SETTING␣UP"
110 LN=990:READ P,B,D,X:IF P+B+D=X THEN 130
120 PRINT "ERROR␣IN␣999":END
130 FOR Z=2049 TO 2586:S=S+PEEK(Z):NEXT
140 IF S=P THEN 160
150 PRINT "ERROR␣IN␣BASIC␣PROGRAM":END
160 CS=0:FOR Z=1 TO 6:READ V
170 IF V<0 OR V>255 THEN 220
180 POKE B+CB,V:CB=CB+1:CS=CS+V:NEXT
190 PRINT ".";:A=A+1
200 LN=LN+10:READ CK:IF CS=CK THEN 160
210 PRINT:PRINT "ERROR␣IN␣LINE";LN:END
220 PRINT:PRINT:IF V=-1 THEN 250
230 PRINT "ILLEGAL␣VALUE";V
240 PRINT "IN␣LINE";LN+10:END
250 IF A=D THEN SYS 51000:NEW
260 PRINT "WRONG␣NUMBER␣OF␣DATA␣LINES":END
999 DATA 38661,51000,61,89722
1000 DATA 076,204,199,169,000,133,781
1010 DATA 253,133,254,133,251,133,1157
1020 DATA 252,165,043,133,176,165,934
1030 DATA 044,133,177,160,000,177,691
1040 DATA 176,133,163,200,177,176,1025
1050 DATA 133,164,240,041,200,177,955
1060 DATA 176,141,053,003,200,177,750
1070 DATA 176,141,054,003,173,053,600
1080 DATA 003,032,126,200,173,054,588
1090 DATA 003,032,126,200,230,253,844
1100 DATA 208,002,230,254,165,163,1022
1110 DATA 133,176,165,164,133,177,948
1120 DATA 076,077,199,169,013,032,566
1130 DATA 210,255,169,018,032,210,894
1140 DATA 255,166,253,165,254,032,1125
1150 DATA 205,189,160,000,185,180,919
1160 DATA 199,240,007,032,210,255,943
1170 DATA 200,076,150,199,166,251,1042
1180 DATA 165,252,032,205,189,169,1012
1190 DATA 013,032,210,255,032,210,752
1200 DATA 255,108,002,003,032,076,476
1210 DATA 073,078,069,083,044,032,379
1220 DATA 080,082,079,079,070,032,422
1230 DATA 078,085,077,066,069,082,457
```

(continued)

```
1240 DATA 032,061,032,000,169,215,509
1250 DATA 141,004,003,169,199,141,657
1260 DATA 005,003,096,032,124,165,425
1270 DATA 186,189,001,001,201,161,739
1280 DATA 240,038,173,000,002,201,654
1290 DATA 081,240,013,201,080,208,823
1300 DATA 008,173,001,002,208,003,395
1310 DATA 076,059,199,096,173,001,604
1320 DATA 002,208,250,169,124,141,894
1330 DATA 004,003,169,165,141,005,487
1340 DATA 003,108,002,003,152,072,340
1350 DATA 169,000,133,251,133,252,938
1360 DATA 165,020,032,126,200,165,708
1370 DATA 021,032,126,200,169,000,548
1380 DATA 133,253,162,000,189,000,737
1390 DATA 002,240,039,201,143,240,865
1400 DATA 035,201,032,240,022,201,731
1410 DATA 034,240,007,032,126,200,639
1420 DATA 232,076,032,200,072,165,777
1430 DATA 253,073,001,133,253,104,817
1440 DATA 076,049,200,165,253,240,983
1450 DATA 237,169,032,076,049,200,763
1460 DATA 166,214,202,142,052,003,779
1470 DATA 162,000,189,157,200,240,948
1480 DATA 007,032,210,255,232,076,812
1490 DATA 084,200,165,252,166,251,1118
1500 DATA 032,205,189,162,003,169,760
1510 DATA 032,032,210,255,202,016,747
1520 DATA 248,174,052,003,134,214,825
1530 DATA 169,013,032,210,255,104,783
1540 DATA 168,096,141,052,003,160,620
1550 DATA 008,024,014,052,003,038,139
1560 DATA 251,038,252,144,012,165,862
1570 DATA 252,073,016,133,252,165,891
1580 DATA 251,073,033,133,251,136,877
1590 DATA 208,231,096,019,018,058,630
1600 DATA 032,000,000,000,000,000,32
1610 DATA -1
```

UNDERSTANDING THE LISTINGS

The program listings use a special notation that tells you which keys to press, instead of showing the graphics symbols used by Commodore. Before you type any of the programs, take some time and become familiar with the notation. It looks a little strange at first, but once you understand just a few rules, you'll be ready to type the programs.

Our program listings use a special notation to show graphic characters. We selected characters that do not appear on the Commodore 64 keyboard to represent the special notation.

Curly braces ("{" and "}") are used to enclose the special notation, which will be explained below. Important spaces in the listings are shown with a caret, so you'll see exactly where spaces are needed. For example,

```
170 PRINT "THE^SKY^IS^BLUE" :rem 1023
```

has three "important" blanks, each shown with a caret.

The degree symbol indicates repeated characters. For example, if you see

```
200 PRINT "{10°down}" :rem 9026
```

you should type:

the number 200
the word PRINT
a quote (")
press the cursor-down key 10 times
and the closing quote (")

The example shows two important points about the notation. First, curly braces are never typed, since they indicate that special notation is being used. Second, the degree symbol is never typed. It always comes between a number (the repeat count) and another symbol. The degree symbol is *not* on the C-64 keyboard, which is why we used it to tell you how many times to press the following key.

In the listings, long lines are printed as several lines in the book, even though you will type them as one long "wraparound" line on the C-64. We always break the line at a convenient point selected to "make sense" when typing the listing. You should only press RETURN at the very end of each BASIC line, regardless of how many printed lines are shown in the book. Press RETURN just before the ":rem" which sets off the PROOF-IT proof number at the end of each line.

In the listings we use lowercase names such as {up} and {red} for cursor and color keys. Graphics are shown as the actual key to press (usually a capital letter) inside curly braces. Table 1 shows how the keys appear in the listings, the key (or keys) you must press, and a picture of the graphic symbol that will appear on the C-64 screen. A few minutes spent studying Table 1 before you type your first program may save you quite a bit of time.

Cursor and Function Keys

In the listings, the cursor keys are shown as {up}, {down}, {left}, {right}, {home}, and {clr}, while {inst} means "press the insert key." These are always inside quotes, since otherwise they would actually move the cursor around on the screen. A few programs use C-64 function keys, shown as the name of the key, such as {f1} for the F1 function key.

You move the cursor on the C-64 screen by pressing the two cursor keys (they are labeled "CRSR," one with left and right arrows, the other with up and down arrows). To move the cursor in a program, we use cursor characters in a string, which is always inside quotes.

Things get a little complicated when you are dealing with cursor keys inside of quotes. When you type a program, the C-64 knows you are in "quote mode" when you press a double quote (the shifted 2). You remain in quote mode until you type the closing quote. Once you are in quote mode, the only cursor movement key that "works" (instead of being placed into the string) is the delete function DEL. So, you can back up and correct mistakes with DEL, but you can't move back and forth with the arrow keys. If you get confused, you can "bail out" by hitting RETURN, which ends "quote mode." Then, go back up on the line and correct it as needed (and check the PROOF-IT number when you press RETURN).

The Color Keys

The C-64 supports 16 colors. For the first eight color keys, we use the names printed on the front of the number keys. When you see a color name in curly braces, press the CTRL key, hold it down, and press the number key with that name. For example, when you see

```
150 PRINT "{red}HELLO" :rem 41850
```

type:

the number 150
a blank
the word PRINT
a quote (")
hold down CTRL and press 3 (for red)
the word HELLO
and a closing quote (")

This one-line program prints the word "HELLO" in the color red.

We show the second set of eight colors (produced by pressing the COMMODORE key, holding it, and pressing one of the number keys) as the color printed on the key, with underlining to indicate the COMMODORE key. Whenever you see something underlined in our listings it means "hold down the COMMODORE key and press the underlined key." For example, {blk} produces orange, the second color for the "blk" key. Look at Table 1 for a full list of the underlined color names, and the keys they represent.

Table 1. Notation for Program Listings

Listed:	Key(s) to Press:	Screen Graphics:	Listed:	Key(s) to Press:	Screen Graphics:
{home}	HOME		{blu}	COMMODORE 7	
{clr}	SHIFT CLR		{yel}	COMMODORE 8	
{up}	SHIFT CRSR UP		{rvs-on}	CTRL 9	
{down}	CRSR DOWN		{rvs-off}	CTRL 0	
{left}	SHIFT CRSR LEFT		{pi}	SHIFT ↑	
{right}	CRSR RIGHT		{space}	SPACE BAR	
{inst}	SHIFT INST		{shift space}	SHIFT SPACE BAR	
{blk}	CTRL 1		{A}	SHIFT A	
{wht}	CTRL 2		{B}	SHIFT B	
{red}	CTRL 3		{C}	SHIFT C	
{cyn}	CTRL 4		{D}	SHIFT D	
{pur}	CTRL 5		{E}	SHIFT E	
{grn}	CTRL 6		{F}	SHIFT F	
{blu}	CTRL 7		{G}	SHIFT G	
{yel}	CTRL 8		{H}	SHIFT H	
{blk}	COMMODORE 1		{I}	SHIFT I	
{wht}	COMMODORE 2		{J}	SHIFT J	
{red}	COMMODORE 3		{K}	SHIFT K	
{cyn}	COMMODORE 4		{L}	SHIFT L	
{pur}	COMMODORE 5		{M}	SHIFT M	
{grn}	COMMODORE 6		{N}	SHIFT N	

Listed:	Key(s) to Press:	Screen Graphics:	Listed:	Key(s) to Press:	Screen Graphics:
{O}	SHIFT O		{J}	COMMODORE J	
{P}	SHIFT P		{K}	COMMODORE K	
{Q}	SHIFT Q		{L}	COMMODORE L	
{R}	SHIFT R		{M}	COMMODORE M	
{S}	SHIFT S		{N}	COMMODORE N	
{T}	SHIFT T		{O}	COMMODORE O	
{U}	SHIFT U		{P}	COMMODORE P	
{V}	SHIFT V		{Q}	COMMODORE Q	
{W}	SHIFT W		{R}	COMMODORE R	
{X}	SHIFT X		{S}	COMMODORE S	
{Y}	SHIFT Y		{T}	COMMODORE T	
{Z}	SHIFT Z		{U}	COMMODORE U	
{A}	COMMODORE A		{V}	COMMODORE V	
{B}	COMMODORE B		{W}	COMMODORE W	
{C}	COMMODORE C		{X}	COMMODORE X	
{D}	COMMODORE D		{Y}	COMMODORE Y	
{E}	COMMODORE E		{Z}	COMMODORE Z	
{F}	COMMODORE F		{+}	SHIFT +	
{G}	COMMODORE G		{±}	COMMODORE +	
{H}	COMMODORE H		{−}	SHIFT −	
{I}	COMMODORE I		{–}	COMMODORE −	
{*}	SHIFT *		{£}	COMMODORE £	
{*}	COMMODORE *		{@}	SHIFT @	
{£}	SHIFT £		{@}	COMMODORE @	

The SHIFT and COMMODORE Keys

The Commodore 64 uses most keys for three different codes. All of the letter keys (and a small handful of other keys) produce a letter and two different graphic symbols. The graphic printed on the right side of the front of a key is produced by holding SHIFT and pressing the key. The graphic on the left front of a key is produced by holding the COMMODORE key and pressing the letter key.

In the listings we show SHIFT graphics as a capital letter inside curly braces. For example, {Z} means "hold down SHIFT, and press the letter Z." Besides the 26 letters of the alphabet, the commercial "at" sign, the asterisk, the plus and minus sign, and the "pound" sign sometimes appear inside curly braces, since they produce Commodore graphic characters.

The rule is simple: capital letters inside curly braces indicate SHIFTed graphic characters, while underlined capital letters inside braces mean COMMODORE graphic characters. A common mistake is forgetting to hold down the SHIFT or COMMODORE keys for characters inside curly braces. If a PROOF-IT number doesn't match, chances are you've forgotten to hold down the COMMODORE key for an underlined letter inside curly braces.

Reverse-video and Spaces

Reverse-video is shown in the listings as {rvs-on} and {rvs-off}. These are produced by holding the CTRL key down, and pressing 9 for {rvs-on} and 0 for {rvs-off}.

A long sequence of blanks is fairly common in C-64 programs. We use the repeat notation when more than three important blanks occur together. For smaller numbers of blanks, we use the caret.

Note that blanks inside curly braces are *not* typed, unless you see the word {space}. Blanks inside braces are used to separate the spelled-out notation, and are included strictly for ease of reading. For example,

```
130 X$="{2°down 2°right}" :rem 6514
```

means you should type:

the number 130
a blank
the letter X
a dollar sign
an equal sign
a quote
cursor-down
cursor-down
cursor-right
cursor-right
a quote.

Notice that that *no* spaces were typed inside the quotes in this example.

The blanks inside braces are strictly for easy reading. When a blank is required, it will be shown as a caret ("^") outside of curly braces, or the word {space} inside braces.

THE STANDARD FRAMEWORK

All programs (except those named QUICK) use a standard framework of lines from 60000 to 62200. Since this section is always the same, you should type it once, **SAVE** it on disk or tape, and then always **LOAD** it right after you **LOAD** and **RUN** the PROOF-IT program. That way, you will know that lines 60000-62200 are always right, and will save lots of typing.

There is a reminder at the end of each program that uses the framework. However, you should **LOAD** the framework *before* you start typing your program. If you forget to do so, but have a floppy disk, the MERGE program will merge the program you typed together with the framework, and create a new file. However, **LOAD**ing the framework before you begin typing a new program is better, since that is much faster than using MERGE. If you use cassette tape, and forget to **LOAD** the standard framework first, you will have to go ahead and type it again. Since that's a lot of extra work, remember to **LOAD** the framework before you start typing!

Getting Started with the Standard Framework

An important thing to understand about the standard framework is that it is *not* a program that you'll ever **RUN** by itself. It is a set of routines that we use from our programs with **GOSUB** or **GOTO** commands. Follow these steps:

1. **LOAD** the PROOF-IT program, and **RUN** it. This lets you proofread each line of the framework as you type it in.
2. Type the standard framework.
3. Press P followed by RETURN. Check the number of lines and program proof number against the listing, and make sure they match.
4. **SAVE** the standard framework (we call it "FRAME") on disk or tape.

After you have a good version of the framework, follow this procedure when you want to type a new program:

1. **LOAD** the PROOF-IT program, and type "RUN."
2. **LOAD** the standard framework.
3. Type the program, and check the proof number for each line.
4. After typing the entire program, press P and RETURN, and check the number of lines and the total proof number against the listing. If both numbers match, the program is correct.
5. **SAVE** the program you have just typed on disk or tape *before* you **RUN** it. There's always some slight chance that something might happen, and if you haven't **SAVE**d a copy you would have to retype it.

```
60000 IN$="":ZT=TI:ZC=2:ZD$=CHR$(20) :rem 2829
60010 GET Z$:IF Z$<>"" THEN 60070 :rem 37749
60020 IF ZT<=TI THEN PRINT MID$("^{+}",ZC,1);"{left}";:ZC=3-ZC:
      ZT=TI+15 :rem 21796
60030 GOTO 60010 :rem 58145
60070 Z=ASC(Z$):ZL=LEN(IN$):IF (Z AND 127)<32 THEN PRINT "^{left}";:
      GOTO 60110 :rem 31223
60090 IF ZL>=QI THEN 60010 :rem 60456
60100 IN$=IN$+Z$:PRINT Z$;ZD$;Z$; :rem 54433
60110 IF Z=13 THEN PRINT CR$;:RETURN :rem 22243
60120 IF Z=20 AND ZL>0 THEN IN$=LEFT$(IN$,ZL-1):PRINT "{left}";:
      GOTO 60010 :rem 20145
60130 IF Z=141 THEN Z$=CHR$(-20*(ZL>0)):FOR Z=1 TO ZL:PRINT Z$;:NEXT:
      GOTO 60000 :rem 51708
60140 GOTO 60010 :rem 42693
60200 ZJ=TI+30:ZT=TI:ZS=2:PRINT LEFT$(JC$,1);PR$;"{up}" :rem 42174
60210 IF (PEEK(JS) AND 16)=0 THEN ZS=2:GOSUB 60280:PRINT:RETURN :rem 64597
60215 GOSUB 60500 :rem 61654
60220 IF TI>=ZT THEN GOSUB 60280:ZT=TI+15:ZS=3-ZS :rem 122
60230 Z=PEEK(JS) AND 12:IF Z=12 THEN ZJ=0:GOTO 60210 :rem 25639
60240 IF TI<ZJ THEN 60210 :rem 4748
60250 IF Z=4 AND IN<JM THEN IN=IN+1:GOTO 60200 :rem 11785
60260 IF Z=8 AND IN>1 THEN IN=IN-1:GOTO 60200 :rem 39353
60270 GOTO 60210 :rem 29622
60280 PRINT TAB(JT+JW*(IN-1));MID$(JC$,ZS,1);MID$(PR$,JT+JW*(IN-1)+1,JW);
      "{up}" :rem 18833
60290 RETURN :rem 23108
60500 GET Z$:IF Z$<>"Q" THEN RETURN :rem 46928
60600 GET Z$:IF Z$<>"" THEN 60600 :rem 33729
60605 POKE VIC+24,21:POKE VIC+21,0:PRINT CHR$(9); :rem 33882
60610 GOSUB 61000:POKE VIC+33,6:POKE VIC+32,14:POKE SID+24,0:
      PRINT "{clr blu}":END :rem 64373
61000 CRT=1024:VIC=53248:WD=40:CR$=CHR$(13):SID=54272:JS=56320:
      CM=55296 :rem 14404
61010 JC$="{blu yel}":QL=214:QI=255:RETURN :rem 7983
62000 L0=LEN(PG$)+2:L1=LEN(AU$)+2:L2=LEN(A2$)+2:IF L1<L2 THEN
      L1=L2 :rem 41621
62010 IF L0<L1+2 THEN L0=L1+2 :rem 35291
62020 B0$=LEFT$("{blu rvs-on 39°space}",L0+2) :rem 58433
62030 DEF FNT(N)=(40-N)/2:B1$="{grn}"+MID$(B0$,2,L1+1):T1=FNT(L1):
      T0=FNT(L0) :rem 14820
62035 GOSUB 61000:POKE VIC+32,0:POKE VIC+33,0:POKE SID+24,0 :rem 45081
62040 PRINT "{clr 3°down}";CHR$(8):FOR I=1 TO 4:PRINT TAB(T0);B0$:
      NEXT I :rem 50139
62050 PRINT "{3°up wht}";TAB(FNT(LEN(PG$)));PG$:PRINT :rem 10983
62060 FOR I=1 TO 3:PRINT TAB(T1);B1$:NEXT I :rem 59320
62070 PRINT "{2°up rvs-on}";TAB(FNT(LEN(AU$)));AU$ :rem 29924
```

(Standard framework continued on next page)

```
62080 IF A2$<>"" THEN PRINT "{rvs-on}";TAB(FNT(LEN(A2$)));A2$:
      PRINT TAB(T1);B1$ :rem 64057
62090 BG$="PRESS␣"+BG$+"␣TO␣BEGIN":T0=FNT(LEN(BG$)+2) :rem 41857
62100 LN$=LEFT$("{40°*}",LEN(BG$)) :rem 49151
62110 PRINT TAB(T0);"{2°down red down - up left A}";LN$;"{S down left -}"
      :rem 10667
62120 PRINT TAB(T0);"{Z}";LN$;"{X}" :rem 38751
62140 PRINT TAB(3);"{4°down pur}COPYRIGHT␣(C)␣1984␣␣THE␣CODE␣WORKS"
      :rem 10843
62150 L1=LEN(BG$):I=1:PRINT "{8°up}" :rem 29706
62160 PRINT SPC(T0+1);MID$("{pur cyn}",I,1);LEFT$(BG$,L1);"{up}" :rem 22657
62170 L1=L1+1:IF L1>LEN(BG$) THEN L1=1:I=3-I :rem 10665
62180 GET T$:IF T$<>"" THEN 62200 :rem 11382
62190 IF PEEK(JS) AND 16 THEN 62160 :rem 29993
62200 CLR:GOSUB 61000:PRINT "{clr wht}":GOTO 100 :rem 50577

      49 lines, proof number = 15502
```

Important Variables in the FRAMEWORK

AU$ First author's name
A2$ Second author's name
BG$ Prompt message for what to press to exit title page
CM Base address of color memory
CRT Base address of screen memory
CR$ Character code of a carriage return
IN Default and final choice for the joystick input routine
IN$ Return string of the input routine
JC$ For the joystick input routine, holds two color code characters. The first is for choices not selected, the second is for the selected choice.
JM Number of options in the joystick input routine
JS Address of joystick port number 2
JT Tab position of first choice for joystick input routine
JW Width in characters of each choice for joystick input routine
PG$ Program title
PR$ Prompt line for the joystick input routine
QI Maximum number of characters for the keyboard input routine
QL Address of current line number on screen
SID Base address of SID sound chip
VIC Base address of the VIC chip
WD Width of screen (in columns)
Z ASCII code of key-press and temporary joystick value
Z$ Current key pressed
ZC Cursor on/off toggle flag
ZD$ Character code of DEL key
ZJ Limits speed of changing choice in joystick input
ZL Current length of input string for the keyboard input routine
ZS Current color code of blinking option for the joystick input routine
ZT Timing variable to blink cursor in the keyboard input routine

How the FRAMEWORK Works

60000-60140 Keyboard input routine
60200-60280 Joystick input routine
60500 Checks for "Q"uit key press to end a program
60600-60610 Reset Commodore 64 to standard condition
61000-61010 Defines framework variables
62000-62200 The framework title page

JOYSTICKS

Many of the games in this book use a joystick. At the end of the directions, you'll find a note such as "Uses joystick." Also, the title screen of the program will ask you to press the joystick button to continue.

Plug the joystick into control port 2. We use control port 2 because under some circumstances, port 1 can interfere with the keyboard. If the joystick does not respond, you may have plugged it into the wrong port, or it may be loose. it is easy to accidentally pull the joystick cord so that it does not have good contact. A plastic cassette case placed under the joystick plug is just the right height to relieve the strain, and help the joystick work reliably. Many joysticks are sold for the C-64. We use the regular Atari joysticks (used with the Atari 2600 game console). These joysticks are inexpensive, reliable, and widely available. Some joysticks only let you move in four directions. This style of joystick will work with many of our games, but not with ATTACK, where you frequently need to fire or move diagonally.

OBTAINING PROGRAMS ON DISK

You can purchase a "Starter Pack" containing only PROOF-IT and the Standard Framework on cassette or disk for $8.00. Or, you can get a disk with all the programs in the book for $30.00. Prices include shipping in the United States and Canada. If you buy the disk, you will still need this book for instructions and program listings. Order from:

C-64 Fun and Games, Vol. 2
c/o The Code Works
Box 6905
Santa Barbara, CA 93160 USA